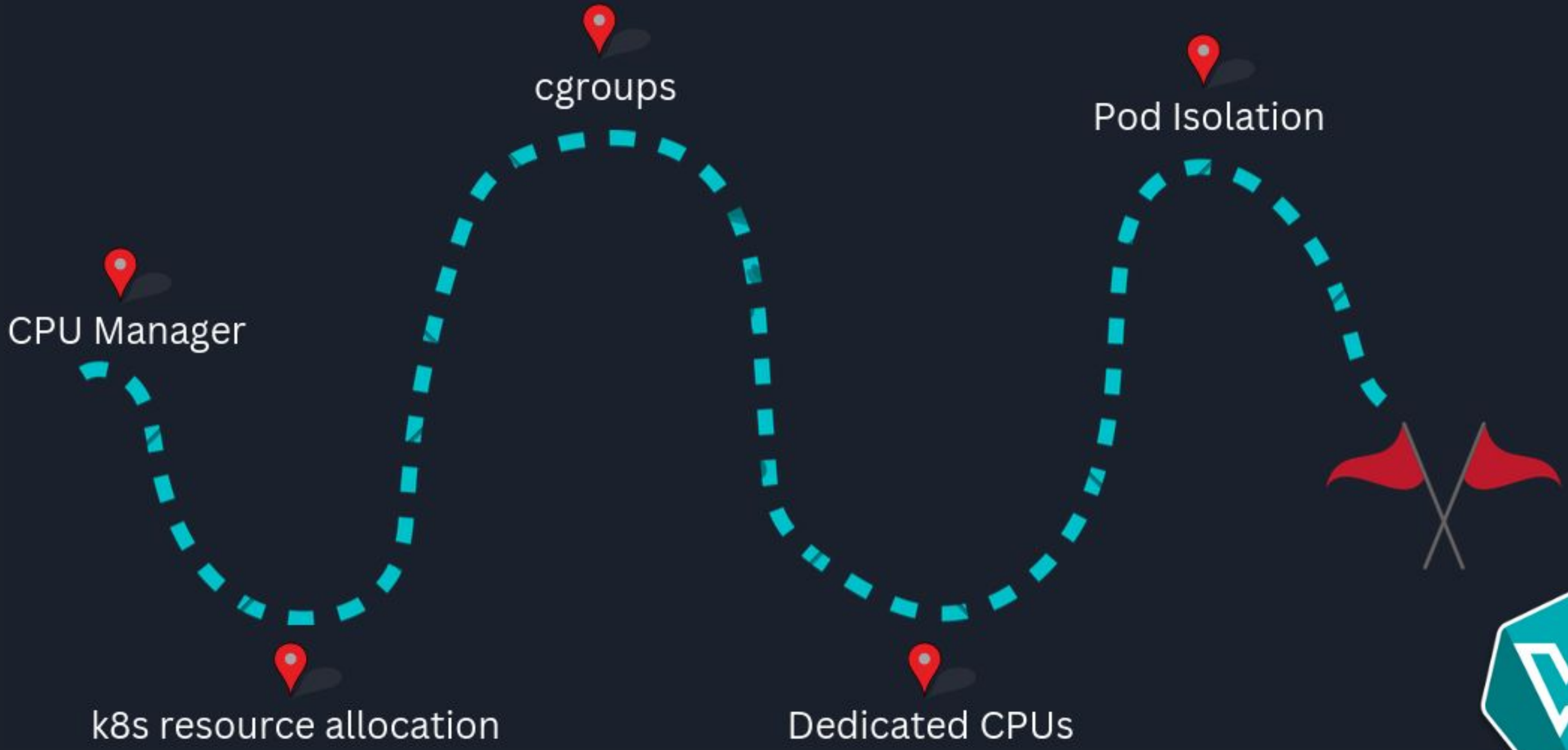
A *journey* through supporting VMs with dedicated CPUs on Kubernetes

Itamar Holder
Senior Software Engineer @ Red Hat

Email: iholder.b@outlook.com
Github: https://github.com/iholder101
Youtube: https://www.youtube.com/@itamarholder8000
Linked-in: https://www.linkedin.com/in/itamar-holder-39095b108/

FOSDEM 2023

A ***journey*** to ...


CPU Manager
k8s resource allocation
cgroups
Dedicated CPUs
Pod Isolation

Introduction to Kubevirt

- Kuberentes + VMs == ***Kubevirt***
- The trick: VM inside a Container

VMs with dedicated CPUs

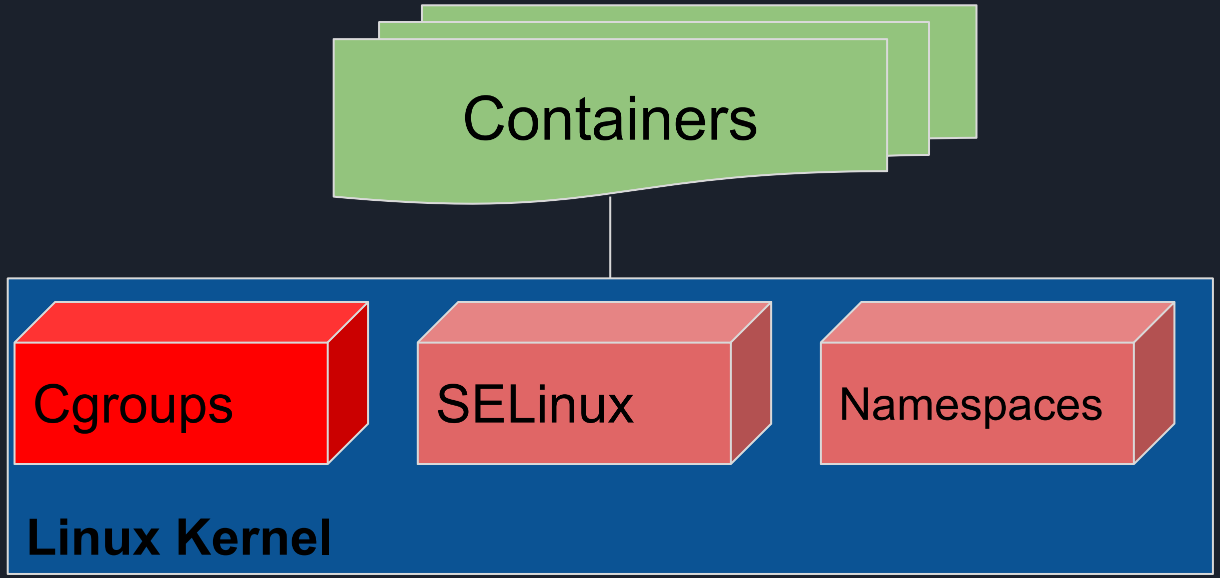
- Crucial for certain use-cases
  - Realtime VMs
  - VMs that depend on low latency
- Supported by most hypervisors
- We aim to bring this support to Kubernetes

Does this look familiar?

```yaml
resources:
  requests:
    cpu: 100m
    ephemeral-storage: 50M
    memory: 1024M
  limits:
    cpu: 200m
    memory: 2048M
```

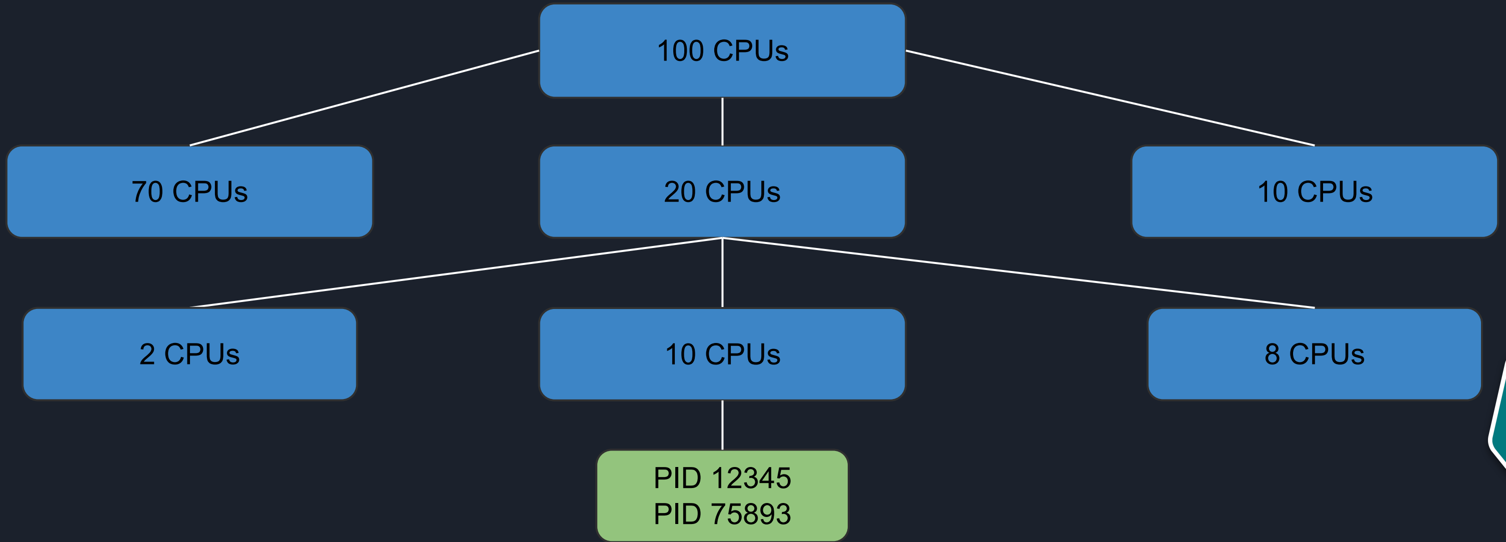
Introduction to Cgroups

- Containers - conceptual concept
- 3 main building blocks



Introduction to Cgroups

- Architecture: tree of resources
- Resources are split between children
- Process attached to cgroup, limited to its resources
- Kubernetes: 1 cgroup per container

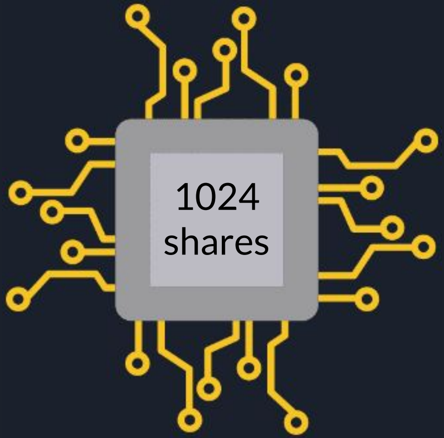


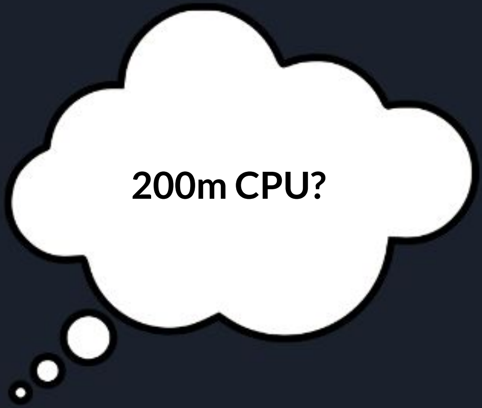
How is CPU allocation implemented in k8s?

	Kubernetes	**Cgroups**
Values are	Absolute	Relative (shares)
Example	100m / 0.1 / 1.3	1024

How k8s converts the absolute values to relative shares?

Kubernetes CPU allocation: requests





- Remember: shares are still relative!
- Side effect: spare resources are available to use
- Request is the ***minimum*** amount allocated

Kubernetes QoS (Quality of Service)

QoS	***CPU Resources***	***Memory Resources***
Best Effort	nil	nil
Burstable	Request: 500m Limit: 1.5	Request: 1024M
Guaranteed	Request: 1.5 Limit: 1.5	Request: 2048M Limit: 2048M

Kubernetes QoS (Quality of Service)

Predictability

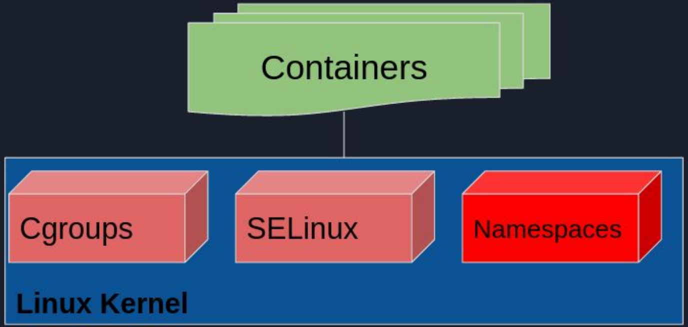
Stability*

* As long as you keep your promises...

Kubernetes and dedicated CPUs

- CPU-Manager => dedicated CPUs on k8s
- Requirements:
  - Guaranteed QoS
  - CPU request (==limit) as an **integer**
- 1 container in a Pod => valid
  - Pod has to be Guaranteed!

Introduction to Namespaces



Sharing PID namespace in a Pod

- A pod can share PID namespace between containers
- As a side-effect, file-systems are also shared!
- The trick: /proc/<PID>/root/

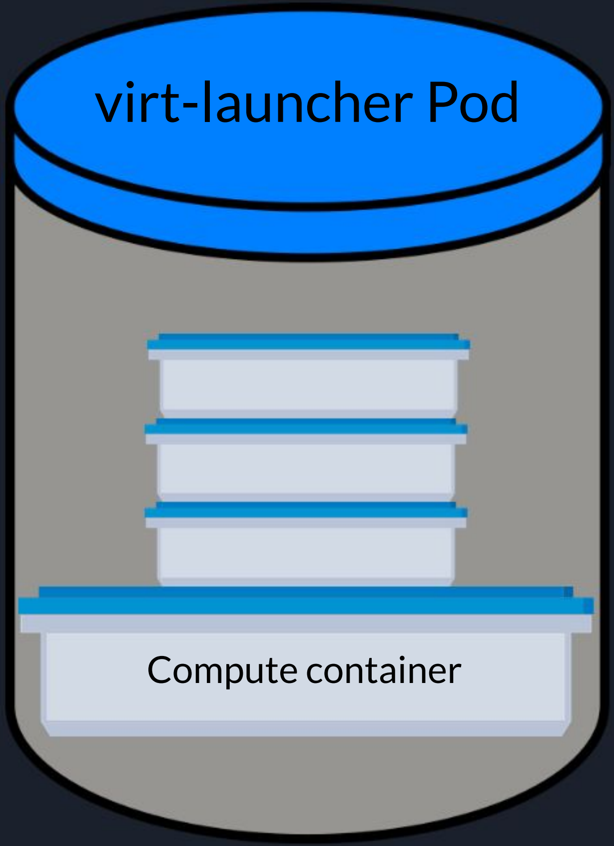
```yaml
kind: Pod
spec:
  shareProcessNamespace: true
```

A word about KVM (Kernel-based VM)

- Kernel module, Linux => Hypervisor
- Kubevirt + KVM == near-to-native performance
- **CPU** virtualization
- Backed by QEMU

A word about KVM (Kernel-based VM)

Back to Kubevirt...



1st attempt to support dedicated CPUs

- The idea: compute container with dedicated CPUs
- Possible with CPU manager
- As long as virt-launcher Pod is of Guaranteed QoS

Inside the compute container

- Many threads, very different priorities
- Most important: **vCPUs**
- Some sibling threads have different priorities

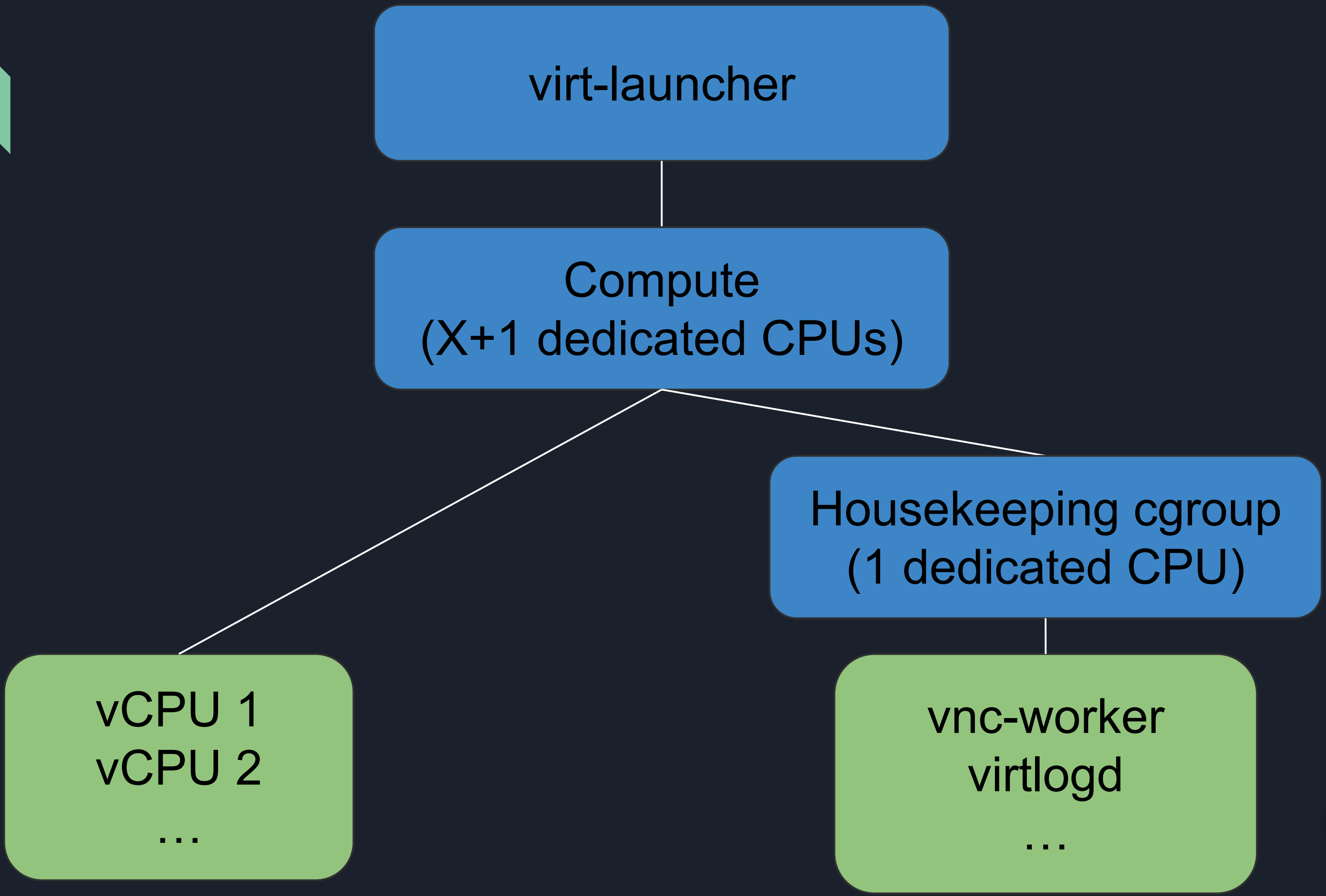
qemu-kvm
CPU 0/KVM
CPU 1/KVM
IO iothread1
IO mon_iothread
vnc_worker
bash
libvirtd
gmain
prio-rpc-libvir
qemu-event
rpc-admin
rpc-libvirtd
vm-default_vmi-
virt-launcher
virt-launcher-m
virtlogd
worker

RED: threads under qemu-kvm process
ORANGE: threads under libvirtd process

Problems with the initial approach

2nd attempt: *housekeeping* approach

- The idea: child cgroup for low-priority threads
  - The ***housekeeping*** cgroup
- User: X CPUs => Allocate: X+1 CPUs
  - 1 (dedicated) for **housekeeping** cgroup
- Move all non-vCPU threads to **housekeeping** cgroup
- => vCPUs with dedicated CPUs

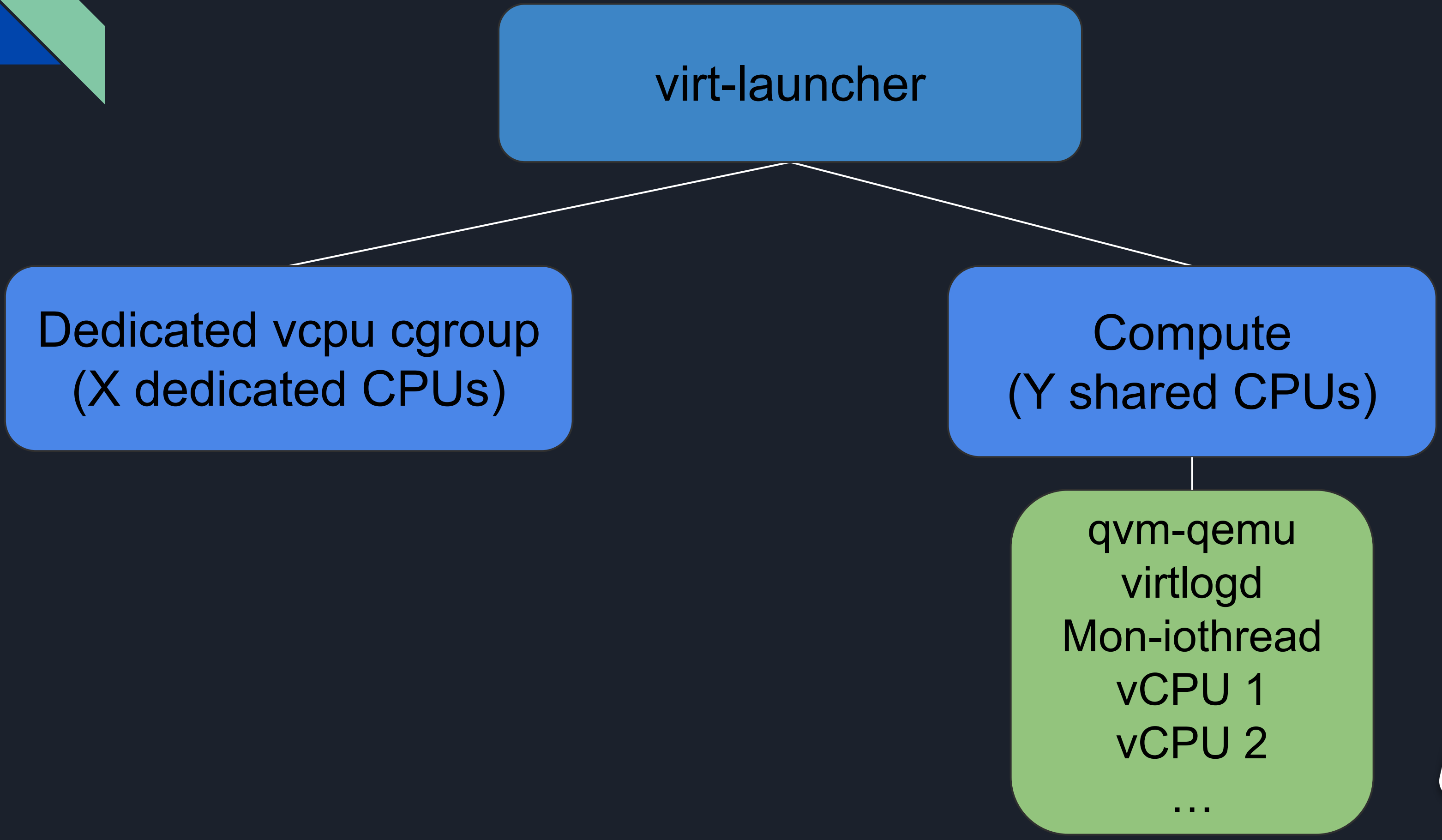

virt-launcher
Compute
(X+1 dedicated CPUs)
Housekeeping cgroup
(1 dedicated CPU)
vCPU 1
vCPU 2
…
vnc-worker
virtlogd
…

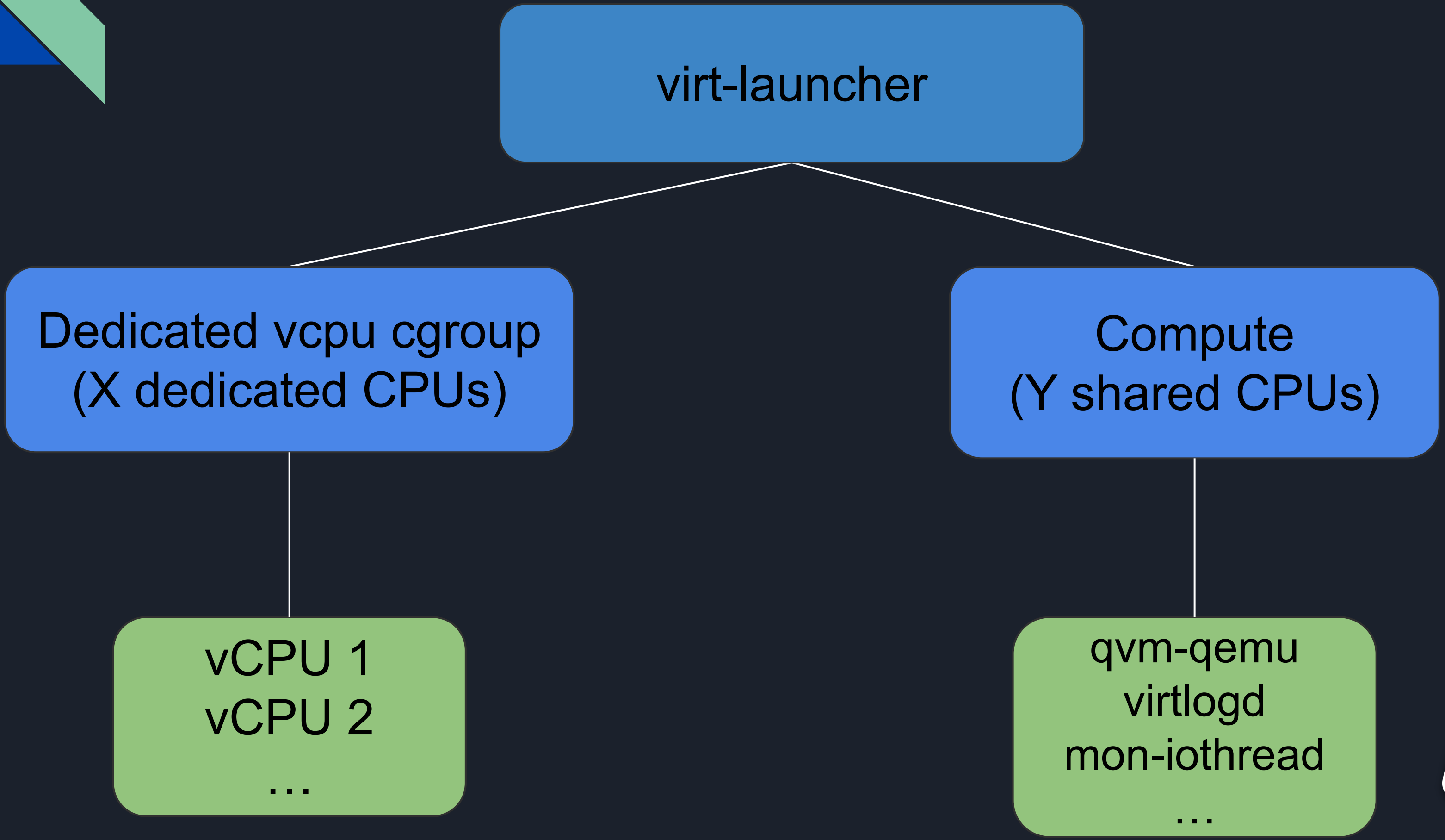
Problems with housekeeping approach

- We **waste 1 dedicated core** that we don't actually need
  - Ideally: X + 0.2 CPUs
  - Impossible in Kubernetes…
- Focused around the lowest priority processes
  - Should be reserved
  - Ideally: **only** configure vCPU threads
- More problems related to cgroups v1/v2
  - Not diving into details here

3rd attempt: *dedicated-cpu cgroup* approach

- Compute container - as usual
  - CPU not dedicated to it
  - Still need Pod Guaranteed QoS
- Instead, new blank container with X dedicated CPUs
  - => new cgroup
- Move only the vCPU threads to this cgroup


virt-launcher
Dedicated vcpu cgroup
(X dedicated CPUs)
Compute
(Y shared CPUs)
qvm-qemu
virtlogd
Mon-iothread
vCPU 1
vCPU 2
…


virt-launcher
Dedicated vcpu cgroup
(X dedicated CPUs)
Compute
(Y shared CPUs)
vCPU 1
vCPU 2
…
qvm-qemu
virtlogd
mon-iothread
…

3rd attempt: *dedicated-cpu cgroup* approach

- Moving threads to another container?
    - Share PID namespace!
- Only relevant threads are being configured
- Shared CPUs for the “housekeeping” tasks
- Avoid allocating extra dedicated core
- Keep things open for extensions in the future

Summary & Takeaways

- A lot of introductions :)
- During our journey, we've seen:
  - CPU allocation implementation in k8s
  - Cgroups
  - Dedicated CPUs on k8s
  - Namespaces + share within a Pod
  - KVM: vCPUs as threads
  - Kubevirt: VMs on k8s
- Hope these takeaway would serve you in one of your journeys

Thank you!

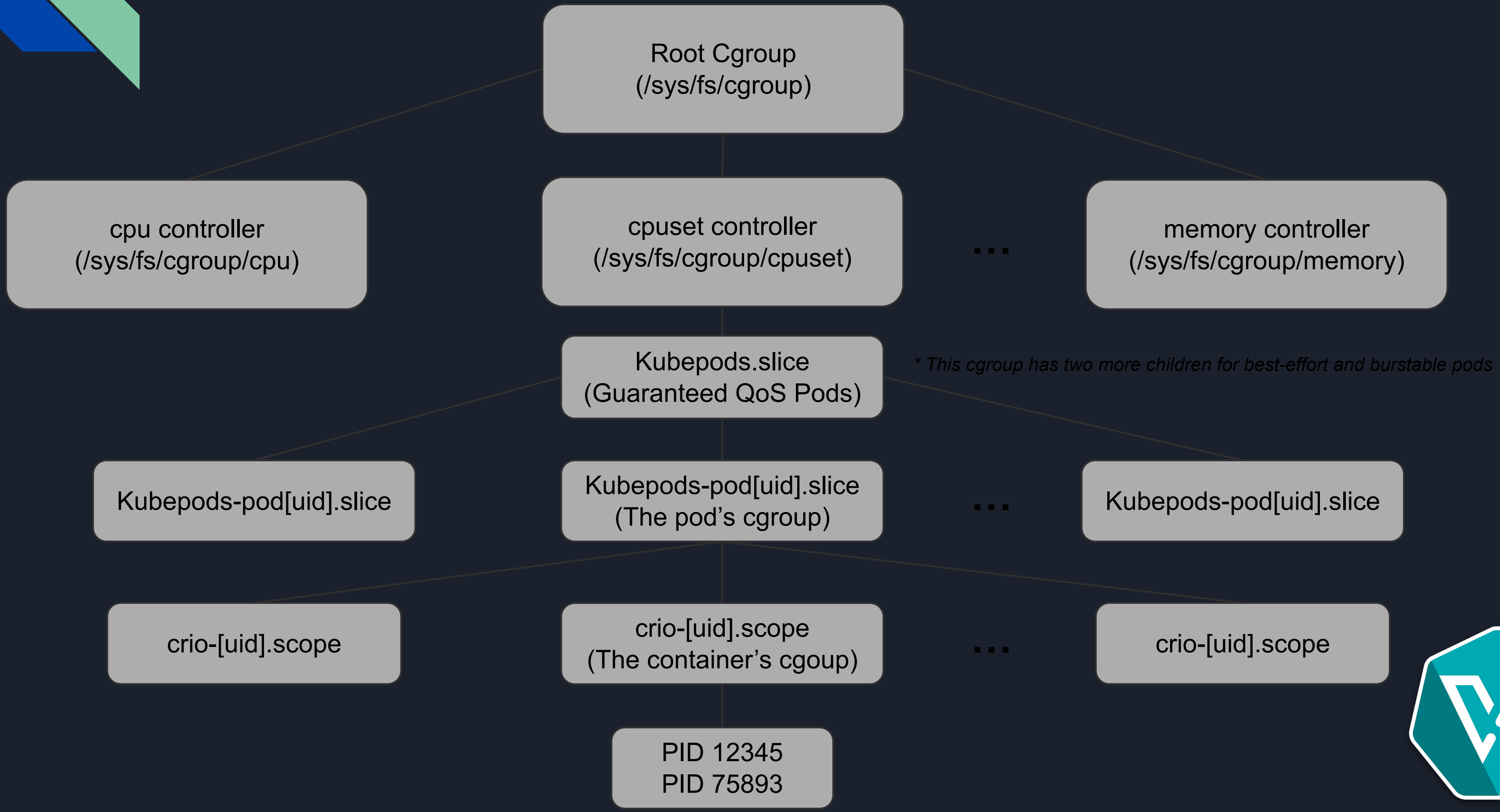
Please feel free to contact me for any further questions!

Itamar Holder
Senior Software Engineer @ Red Hat

Email: iholder.b@outlook.com
Github: https://github.com/iholder101
Youtube: https://www.youtube.com/@itamarholder8000
Linked-in: https://www.linkedin.com/in/itamar-holder-39095b108/

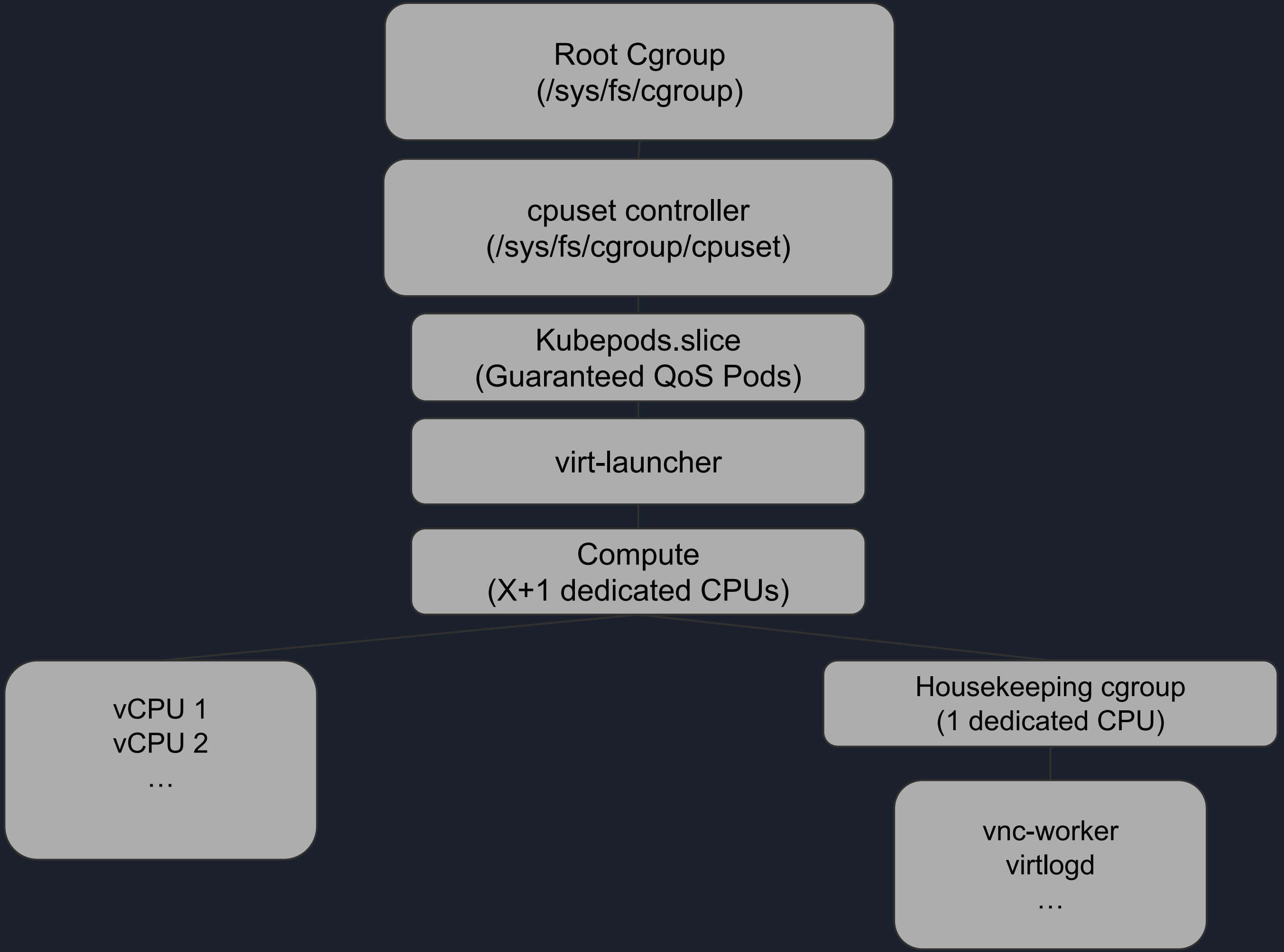
Additional resources

A more detailed cgroup hierarchy in Kubernetes



A word about cgroup v1 / v2

- Cgroups v2 was introduced in March 14th, 2016.
- Cgroup v2 is designed completely different
- No backward compatibility
- Simply put: More restrictions, less error-prone, less generic.
- Both cgroup v1 and v2 are supported in current Linux kernel
- As of today, most workloads still use v1
- GA-ed on Kubernetes 1.25


Root Cgroup
(/sys/fs/cgroup)
cpuset controller
(/sys/fs/cgroup/cpuset)
Kubepods.slice
(Guaranteed QoS Pods)
virt-launcher
Compute
(X+1 dedicated CPUs)
vCPU 1
vCPU 2
…
Housekeeping cgroup
(1 dedicated CPU)
vnc-worker
virtlogd
…